Natural Language Processing
COM3029 & COMM061
Individual Assignment-Group 11

YASH MAHESH KULTHE
URN: - 6766477

I. Group Declaration

1. Data selection:

In this session, and based on the information provided by the professor, we are discussing and choosing 13 labels (included) neutral; So, there are labels we are choosing to do a emotion recognition process:
•Neutral •Surprise •Love •Fear •Joy •Gratitude
•Approval •Disapproval •Confusion •Sadness
•Realization •Desire •Optimism •Pride

2. Experiments:

When planning experiments for emotion recognition data, we have found that there are some necessary steps for doing to reach the target with high accuracy and high collaboration with each member of the team:

a. Define the scope:
   Identify the specific objectives and goals of the sentiment analysis project. Why we do that and the applications of its sentiment analysis. Through it, we can understand the general problem we are currently solving, and it helps us divide the task more efficiently and effectively.
b. Define the data source
c. Define the sample size for training – testing
d. Define the tools and techniques
e. Conduct the sentiment analysis
f. Conclusion the experiments

This multi-class classification approach enables a more comprehensive sentiment recognition by incorporating diverse emotions. The chosen labels cover a wide spectrum of emotions, ranging from positive (e.g., love, joy, gratitude) to negative (e.g., fear, sadness, disapproval), as well as those that are more complex or context-dependent (e.g., surprise, confusion, realization).

Then, it highlights the necessary steps for planning experiments, including defining the scope, data source, sample size, tools, and techniques, emotion recognition, and conclusion of the experiments. The text also mentions the use of GitHub, Google collab, and Dataset as common development environments. Lastly, the individual tasks are separated into data set preparations, algorithms, pre-trained models, setting up hyperparameters and experimental variations, and tracking progress.

This customized 13-label emotion recognition will facilitate a deeper understanding of the underlying emotions present in text data, allowing for more informed decision-making and enhanced communication in various contexts, such as marketing, customer service, product development, and social media monitoring.

COM3029 & COMM061 coursework report

Yash Mahesh Kulthe yk00381@surrey.ac.uk

Abstract

The coursework examines the use of multi-class classifier models for recognizing emotions in text. The GoEmotions dataset, which contains 58k Reddit comments with 27 emotions and a neutral label, was used to conduct experiments on various aspects of the text classification task. The experiments focused on data preprocessing, NLP architectures, text feature extraction, and hyperparameter optimization. The final model used a combination of CNN and BLSTM, which achieved the highest accuracy and outperformed other models. The study emphasizes the importance of careful experimentation and analysis in developing effective models for text classification, and demonstrates the effectiveness of multi-class classifier models in emotion recognition tasks.

1. Data Analysis Visualization

Data Visualization is the most important task in any machine learning/ artificial intelligence task because it gives us insights into the data. For Natural Language Processing (NLP), this is the most important aspect because, we can't simply assume things and give the data to the model for classification tasks as the data might have unstructured data, noise in it, which might affect the performance.

It also allows us to understand the data in a better way and also help us to identify the patterns, complex relationships and trends in the data while analyzing it and pointing out the changes we need to make, or the processes we can or can't work on with.

1.1. Dataset and Visualization

The dataset that we are going to use is the GoEmotions dataset form HuggingFace which contains 58k Reddit comments with 27 emotions and a neutral label. The labels that the group selected has been mentioned in the Group Declaration section of this report.

The dataset selected for this coursework is the 'simplified' dataset which has 58k samples of the text data divided into train, test and validation data. For visualization, only the train data is used for faster execution while in the experiments the datasets are clubbed together and then again splitted for execution.

After dropping the 'id' column, the train_data is copied in visualize so that we don't change the original values of train, test and valid datasets. The number of samples in all the labels can be seen in Figure 1. Then we deleted the labels which we don't need and then label the dataset from 0-13 and visualize it in Figure 2.

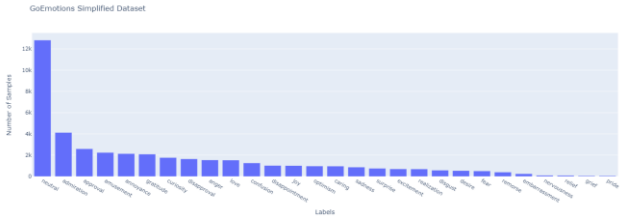


Figure 1: Number of Samples VS Labels Histogram

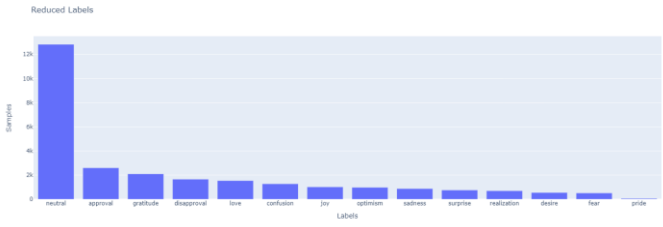


Figure 2: Number of Samples VS Labels Histogram (Selected group labels)

1.2. Text Analysis

a) **Words-per-text**: -

In any NLP task, we need to know if the sentences in the dataset has almost same average length and complexity. In the case of text classification, the length and complexity of the text can be crucial for the performance of the model. If the document has long sentences, it has a lot of information but it is harder to classify due to complexity, while with shorter sentences, it is easy to classify but don't have information for accurate classification.

Words per text have been visualized and shown in Figure 3. In the dataset, the average words per text are around 10-15, which tells that there is not a lot of difference in it.

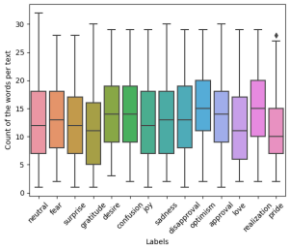


Figure 3: Words per Text VS labels

b) **Length of text**: -

Another main analysis method for NLP is to see the length of texts and number of samples having that length. It helps to determine if we need to process the data in a specific way. If we have very short or very long texts, we will need to normalize it to improve the training.

The visualization of frequency of length of text is shown in Figure 4. A gaussian type distribution can be seen for the dataset. Mostly the dataset contains sentences having around 60 length.

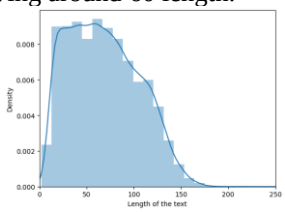


Figure 4: Length of text Frequency

1.3. N-Grams

N-grams is a really powerful technique in NLP it is a representation of text in a sequence of the words of the sentence having a fixed length '*n*'. The n-grams are extracted from the text and the converted n-grams can be used as sequence of features in various NLP tasks.

N-grams not only capture information from the text but also gives us the relationship between the words and can be really good for a model to develop relation between words. N-grams can really improve the accuracy.

N-grams can also be used to information retrieval like the n-grams can be indexed in a matrix, and we can search the same n-gram for our task and we can see if the pair is more frequent or not. In this coursework, Bi-grams and Tri-grams have been used to visualize the relationship between words.

a) **Bigrams**:-

The bi-grams are nothing but just pairs of words appearing successively. For example if we have a sentence- 'I love games', the bigrams would be 'I love', 'love games'.

Bigram can be used to estimate the likelihood of a sequence of words occurring in a given document. Knowing the frequency of each bigram, the model can predict the probability of a bigram which might occur after a piece of text.

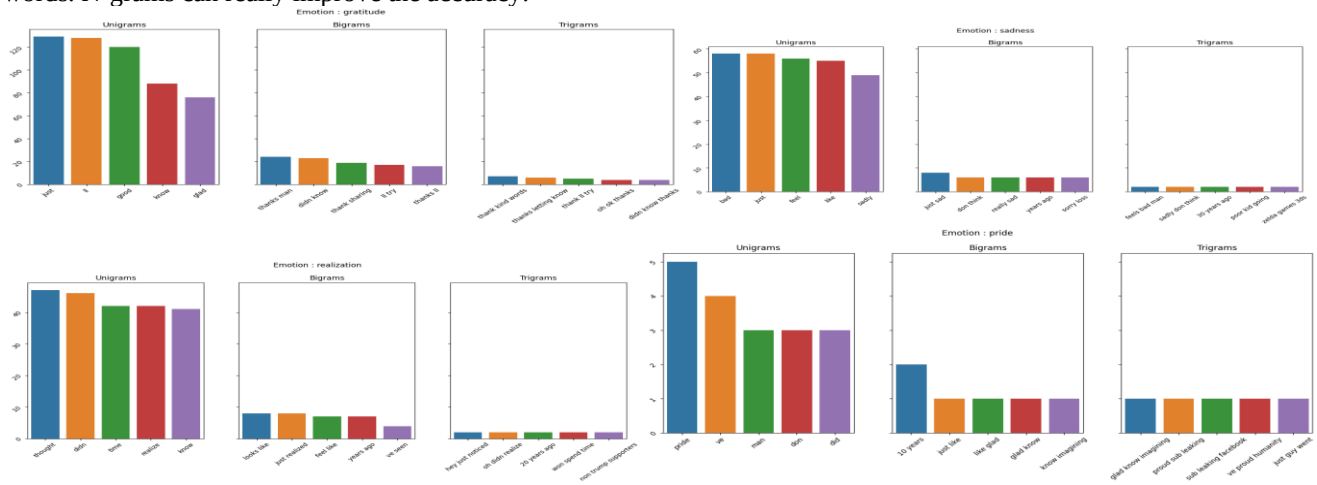
In this analysis part, the top bigrams have been found in each category and visualized to get a better idea of the dataset. The bi-grams can be seen in Figure 5. It can be seen in the pride emotion, that the top frequent bigram is "10 years", having a number in it which is not useful for a model to learn. The numerical values need to be removed as in for the model to learn better.

b) **Tri-grams**: -

Same as bi-grams, tri-grams also are set of words, but this time, it is a sequence of three adjacent words in a sentence. Tri-grams capture more context and information and can tell us better about the relationship between words, better than bigrams.

Tri-grams can be used same as bigram, but they can be more beneficial if we have a model to build text or predict words after some text input.

In this analysis part, the top trigrams have been calculated and visualized which can be seen in Figure 5. Same with the trigrams, there are some numerical values in it as seen in "realization" label, which need to be deleted as they will make our model to predict wrong when unseen data is passed



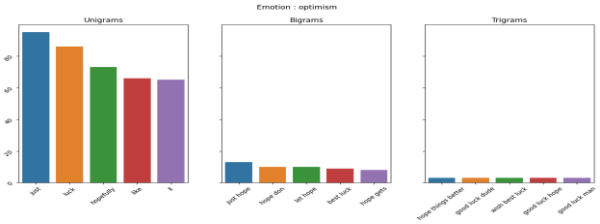


Figure 5: N-grams

2. Flow of the Model

Before going forward with the building of the model, the workflow of the project is shown below in Figure 6

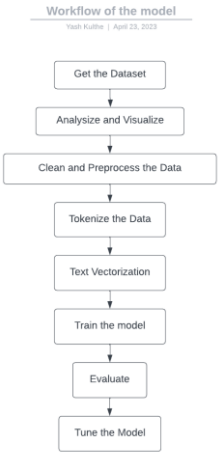


Figure 6: Workflow of the model

3. Helper Functions

For ease of use, there are some helper functions that have been made such that for each experiment is it easier to call them rather than reusing the same code again and again to not make our code too complex and hard to read. The description of each function has been given here itself but not in the experiments section for better referencing.

1) multiple_label_explode: This function is to handle the multi-labelled data to convert it to multi-class data by exploding the multilabels such that the same text which had multilabel will now be duplicated and have those labels as single.

2) hierarchical_label: This function is also used to handle the multi-labelled data by using the hierarchy or setting up ranks for the labels by ourself. Then this function will see all the labels of the multilabeled data and select the one which has higher rank.

3) remove_labels: This function helps to remove the unwanted labels which we don't want from the dataset. It only keeps the 13 labels which were selected.

4) order_grp_labels: This function will map the labels such that it will convert our labels from 0-27 to 0-13.

5) remove_emoji: This function helps in removing the emoji or emoticons from the dataset.

6) remove_unnecessary: This function removes the non-alphanumeric characters.

7) normalization: This function will first convert the dataset to lowercase, then remove punctuations, extra whitespaces, also it will remove the numeric characters which were seen occurring many times in n-grams.

8) undersample: This function will undersample the dataset to a given threshold.

4. Experiment 1 (Data Preprocessing)

The experiment 1 is based on data preprocessing which is key step in any machine learning/artificial intelligence task.

4.1. Generalization

The dataset is being experimented on preprocessing, then it will have same general approach for all the variations: undersample it to 3687 samples (A number taken such that the 4th label is having the same samples which is the second highest after 5th label i.e. 17772.

Then the data is being shuffled with a random seed of 42, we have also kept tensorflow and numpy seeds as 42 at the start of the notebook. Tokenize the data using nltk word_tokenize.

Splitting of the data is followed after this using train_test_split of sklearn, keeping random seed of 42. The dataset is splitted throughout all experiments into 80/10/10 split for train, test and validation respectively

The vectorizer used is TFIDF vectorizer which is used on the tokens. While the labels are kept as numpy array of integers.

SVM model is used with Radial Basis function kernel[1] with gamma = 'scale' and random state of 42. Then the confusion matrix and the classification report is visualized.

4.2. Variation A (Multilabel Handler)

In the GoEmotions dataset, there are 58k samples of text having the labels as emotions. The dataset consists of many samples which are having multilabels i.e. one text having multiple labels like 'gratitude', 'joy', etc. But the problem we working on is a multiclass classification which needs a unique label not multiple labels.

To handle that, we are going to use two methods:

1)multiple_label_explode: With this method, we are going to explode the rows having multilabel such that if a text was having 3 labels, the same text will be duplicated 3 times having those labels as unique value.

2)hierarchical_label: With this method, we have a pre-assigned a dictionary which will be taken as hierarchy each labels have been given a ranking. The function will take in the dataframe, and look for multiple label samples.

After running and evaluating the dataset on the model setting up the parameters given in Generalization(4.1), we got these results, which can be been in Figure 7.

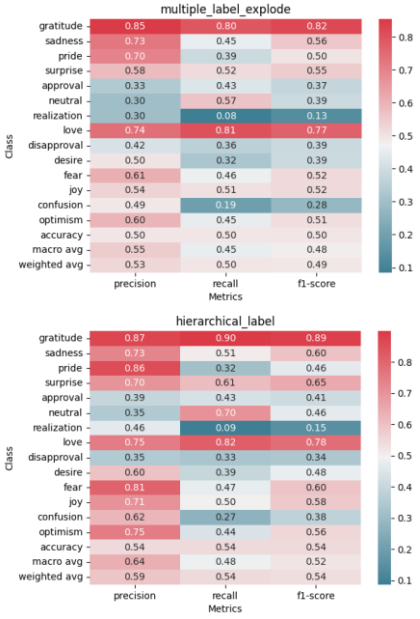


Figure 7: Multilabel Handler (Classification Reports)

The confusion matrix is also a good tool to look at for to see which is a better classification model. The confusion matrix are shown in Figure 8.

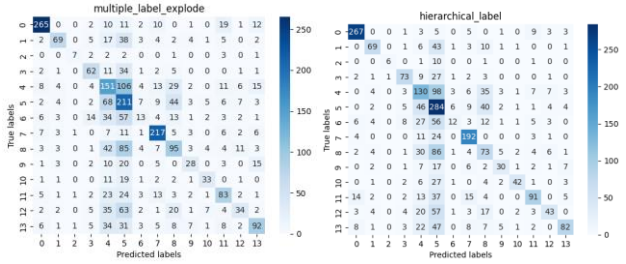


Figure 8: Multilabel Handler (Confusion Matrix)

As with the classification report we can see many f1 scores are better for hierarchical rather than multiple explode also the accuracy for hierarchical label is 54% while for multiple label is 50%

Finally with this we can tell hierarchical labelling is the best as it wont be having different labels for same text as well as we can tune the ranks by our need.

4.3. Variation B (Stemming/Stopwords removal/Lemmatiation)

In this variation, four things were tried:

a) Stemming without stopwords removal
b) Lemmatization without stopwords removal
c) Lemmatization plus stopwords removal
d) Stemming plus stopwords removal

With the all experiments, it was found out that lemmatization is not working well with or without stopwords. This might be due to the dataset as it is a collection of comments from Reddit meaning it won't be having formally written words so to boil it down to the base form word will not work most of the time.

In contrast, stemming works the best as it just gives us the root word, so even if the text is containing words which might not be formally written, it can be converted to the stem word and extract the features.

The results of only stemming with and without stopwords is being showed, for results of lemmatization. The .ipynb notebook[7] can be referred. Confusion matrix can be seen in Figure 9.

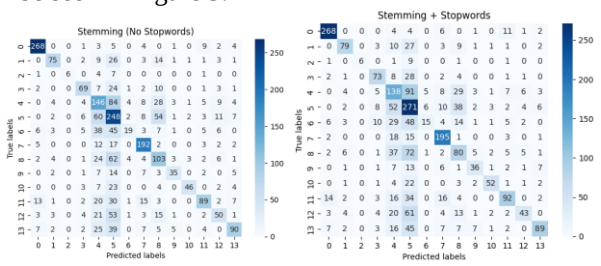


Figure 9: Stemming (Confusion matrix)

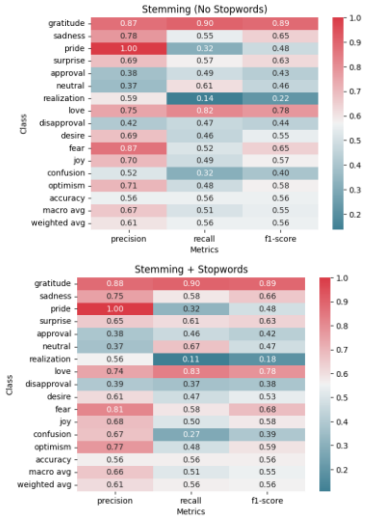


Figure 10: Stemming (Classification report)

4.4. Variation C (N-grams)

N-grams such as bigram and trigram have been tried on the dataset with lemmatization, as using ngrams with stemming doesn't work good as stemming brings down to the root word. Other things are the same as Generalization(4.1).

With the experimentation, it was found that both ngrams are not working well which can be seen in Figure 11. Only classification reports have been shown here. This might be due to as the dataset contains comments from reddit it has slangs, abbreviations and informal grammar, which means that it will take a large feature space as same words are written in different ways. So for this project, we will be using unigrams i.e. tokens only.

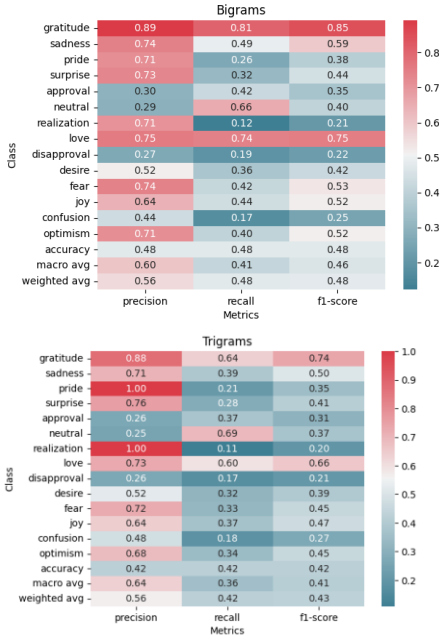


Figure 11: N-grams (Classification report)

4.5. Verdict

For the next experiments, it is best to use the hierarchical label handler as multilabel handler. While using stemming as the main pre processing element. To use stopwords or not depends on the model, while both of the things have been tried but only the best one has been kept. Also, ngrams will not be used as they are producing bad results and the explanation is in 4.4.

The results of SVM experiments can be seen here:

- Multilabel label handler:-
  Validation acc – 0.49
  Test acc – 0.50
- Hierarchical label handler:-
  Validation acc – 0.53
  Test acc – 0.54
- Stemming + stopwords:-
  Validation acc – 0.54
  Test acc – 0.56
- Bi-gram:-
  Validation acc – 0.45
  Test acc – 0.48
- Trigram:-
  Validation acc – 0.39
  Test acc – 0.42

5. Experiment 2 (NLP Architecture)

The experiment 2 is based on NLP Architecture which is the base of this project.

5.1. Generalization

The dataset is being experimented on preprocessing, then it will have same general approach for all the variations: undersample it to 3687 samples. The data is then shuffled with a random seed of 42, we have also kept tensorflow and numpy seeds as 42 at the start of the notebook.

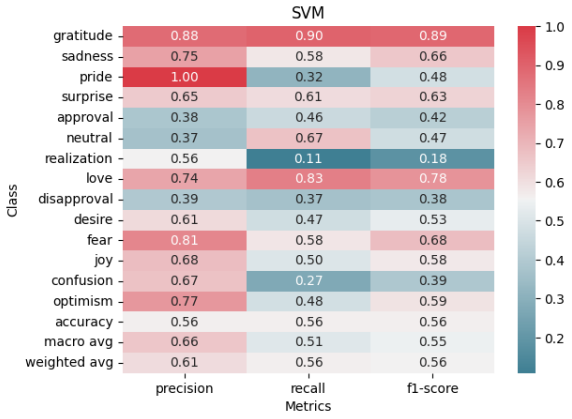
Splitting of the data is followed after this using train_test_split of sklearn, keeping random seed of 42. The dataset is splitted throughout all experiments into 80/10/10 split for train, test and validation respectively

5.2. Variation A (SVM Classifier)

The preprocessed data is first tokenized the same way it was done before using nltk, then splitted into train, test and validation data. Tfidf Vectorizer is being used to fit the vectorizer on the training data and the same vectorizer is used to transform the train, test and validation split.

SVM's have been used for text classification from many years and provide good results for classifications. The Radial Basis Function 'rbf' is used as the kernel [1] keeping random state as 42.

The model is then fit on the train data and evaluated on test data. The results are shown in Figure 12.



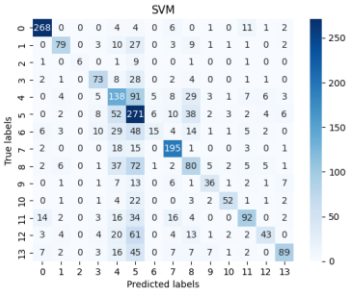


Figure 12: SVM Classifier (Classification report)

5.3. Variation B (LSTM)

In this variation, only stemming is used, stopwords are not being removed as to store the important information which removing the stopwords might cause. The same Generalization(5.1) is used.

The plan in this variation is to use the LSTM model[3], as it has shown many success in the field of NLP.

The labels are then changed to one hot encoding using tensorflow to_categorical function. Then the dataset is being tokenized, tensorflows tokenizer is such that it creates a vocabulary of the words that it is fitted on and gives them a unique word index. Using the word indexes, we can find the vocabulary size which is 12929. One more important step is padding, the text is being padded to 30 as the maximum number of words in a sentence of GoEmotions dataset is 30. Post padding is used which can be seen in Figure 13.

```
[3400, 17, 1005]
[3400   17 1005    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0]
```

Figure 13: Padding applied to a tokenized text.

Next, the LSTM model is defined such that the first layer is the embedding layer. Instead of using any vectorizer, we are using this embedding layer which represents a vector of input data in vectorized format and gives it to the next layer i.e. LSTM. The output dimension is set to 128 while the input length is given same as the length of maximum number of words per text i.e. 30.

Then in LSTM layer, 100 units have been used with a dropout of 0.2 and recurrent dropout of 0.2 to prevent overfitting and perform regularization. Finally a dense layer of 14 output is set with softmax activation.

The optimizer used is adam with default parameters and categorical crossentropy loss function is used as our labels are one hot encoded. The model is trained on 30 epochs and batch size 512. The results are shown in Figure 14.

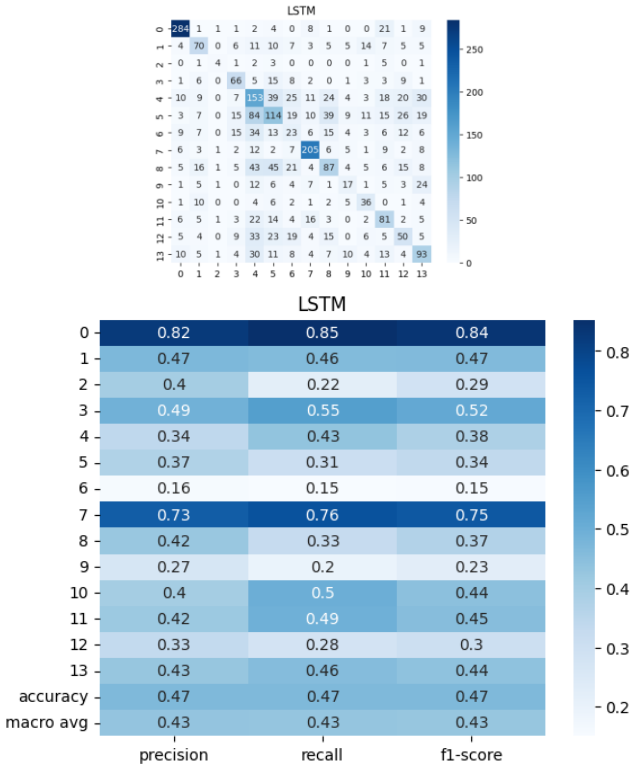


Figure 14: LSTM results

While the model is showing 46% test accuracy but the train accuracy was 82% which clearly shows that the model is overfitted.

5.4. Variation C (BLSTM)

This variation consists of same thing but the only change is that in place of LSTM layer, BLSTM layer is used.

The same text and same one hot encoded labels are being used for this with the same hyperparameters. The only change is in the BLSTM layer, as to use GPU in tensorflow LSTM layer these are the parameters [6] we need to use given in Figure 15.

```
model = Sequential()
model.add(Embedding(input_dim=12929, output_dim=128, input_length=maxlen))
model.add(Bidirectional(LSTM(units=100, dropout=0.2, activation= 'tanh',
                             recurrent_activation='sigmoid',use_bias=True,
                             recurrent_dropout=0, unroll=False,
                             kernel_regularizer=regularizers.l2(0.01))))
model.add(Dense(units=14, activation='softmax'))
```

Figure 15: BLSTM parameters

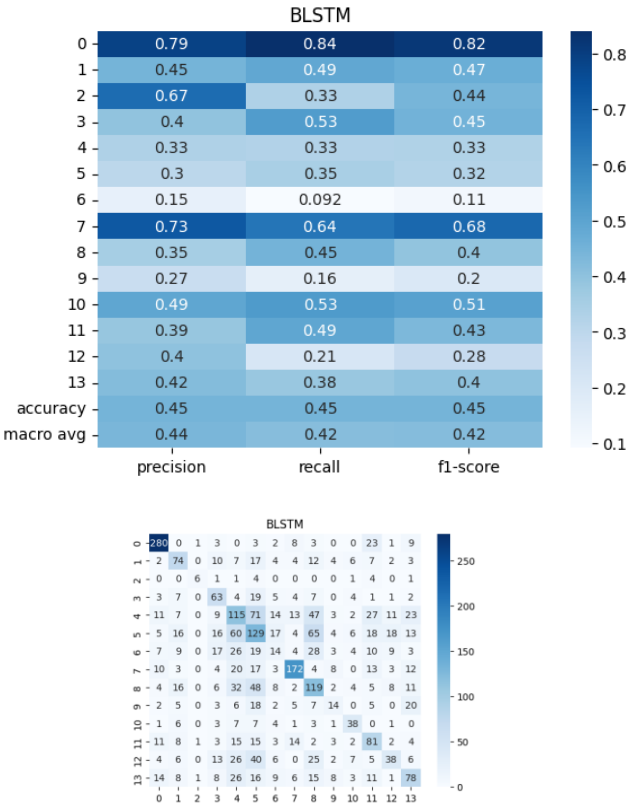


Figure 16: BLSTM results

Figure 16 shows the results of BLSTM layer. The accuracy looks less then LSTM because we have not performed any hyperparameter tuning on it. Also, this model is also overfitting having train accuracy of 83%.

5.5. Verdict

The comparison between the SVM, LSTM and BLSTM on seeing the results, it seems that SVM is performing better with high accuracy of 56%, but also there is one thing that is making this happen that is text vectorization. In SVM, TFIDF is being used, while in LSTM and BLSTM, Embedding layer has been used which learns the vectorization as we train the model.

In the paper[4] it can be seen that LSTM is working better and giving better accuracy i.e 89.1% rather than SVM which is getting 86.4%. So, we will be working on BLSTM as it has functionality to capture information in past and future, i.e. backwards and forwards too. As in emotion classification, context is really important, we are going to use

The results of Algorithm experiments can be seen here:

- SVM:-
  Validation acc – 0.49
  Test acc – 0.56
- LSTM:-
  Train acc – 0.82
  Validation acc – 0.49
  Test acc – 0.46
- BLSTM:-
  Train acc – 0.83
  Validation acc – 0.49
  Test acc – 0.45

6. Experiment 3 (Text Featurization)

The experiment 3 is based on Text featurization which is the key process in building NLP model, without converting text data into data which the model can understand, the model cannot be trained.

6.1. Generalization

The dataset is being experimented on preprocessing, but this time only removal of stopwords is being taken as stemming was tried but it was not giving good accuracy. Then it will have same general approach for all the variations: undersample it to 3687 samples. The data is then shuffled with a random seed of 42, we have also kept tensorflow and numpy seeds as 42 at the start of the notebook.

Splitting of the data is followed after this using train_test_split of sklearn, keeping random seed of 42. The dataset is splitted throughout all experiments into 80/10/10 split for train, test and validation respectively

The changes to be seen here are the vectorization of the text data. Then the same BLSTM model is being used with the same parameters as shown in 5.4. The change here is, the loss function, the labels have been taken as integers, so to work on with that ‘sparse_categorical_crossentropy’ loss is used as categorical crossentropy does not work with integers. Model is trained on 30 epochs and batch size 128

6.2. Variation A (TFiDF instead of Embedding layer)

Previously, an embedding layer was used to train the vectorization, which was pointed out in the Verdict(5.5). To work on that, now the embedding layer is removed but instead of that, TFiDF vectorizer is fit on the train data and the dataset splits are then transformed into matrix of TFiDF features which is learnt from the vocabulary. Then this matrix is compressed using csr_matrix and then make it into array as the model accepts arrays.

The model is then trained on the TDiDF features and the results are obtained as in Figure 17. It is seen that the accuracy has improved a lot, also the overfitting problem has been resolved. The training accuracy is 58% while the testing accuracy being 59%. But with this, it is seen that the 2nd and 6th label are having zero predictions, while all of them are being classified as normal values.

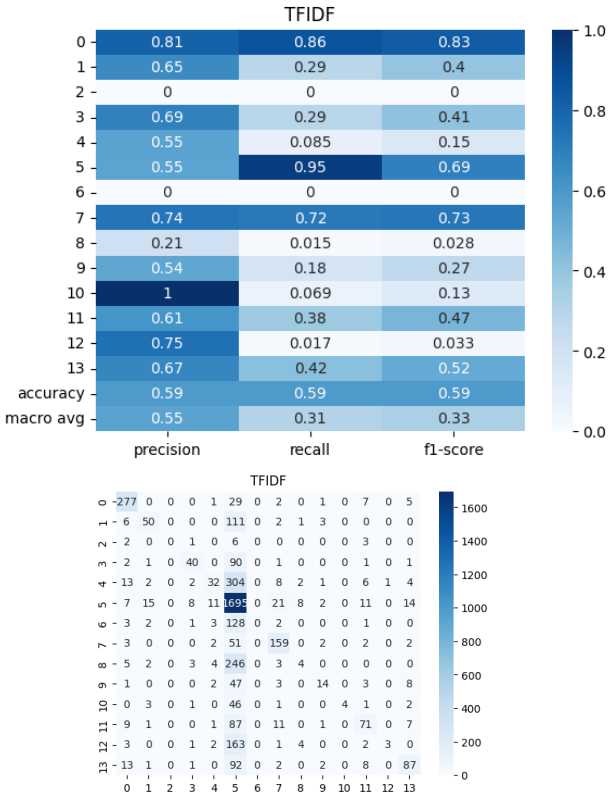


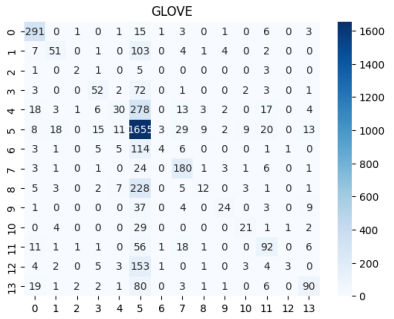
Figure 17: TFiDF on BLSTM results

6.3. Variation B (GloVe Embedding as Embedding layer)

A really good pretrained vectorization method is by using the GloVe embedding. The embedding vocabulary being used in this experiment is the pre-trained embedding model which was trained on large corpus of tweets using GloVe algorithm i.e. glove.twitter.27B.100d.txt. This is chosen because it is trained on tweets which is almost similar data compared to Reddit comments which GoEmotions dataset is.

The GloVe model is extracted and made in a embedding matrix which is then added as pretrained weight to the Embedding layer of the model while making the embedding layers training as false.

The model is trained using the same hyperparameters as last variation. The results are shown in Figure 18.



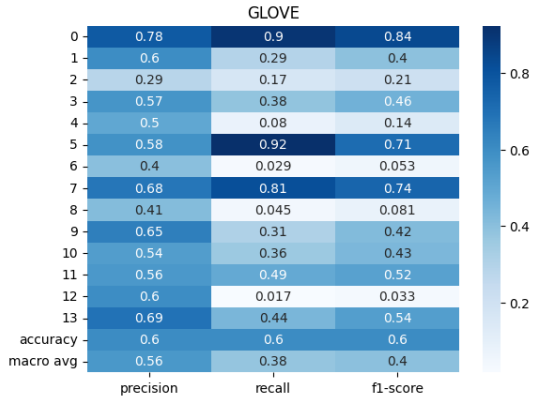


Figure 18: GloVe on BLSTM results

With GloVe embedding the accuracy has been improved also the true predicted classes has also improved in most of the cases.

6.4. Verdict

The problem in last experiment which was of training embedding layer was solved this time using pretrained word embeddings, first TFiDF was used which was fitted in the training data itself while the GloVe embedding is fitted on tweet data of almost 2billion tokens, which is better. The GloVe embedding has shown better accuracy and classification due to it being trained on tweets, so the same embedding will be used and then try to tune it.

The results of Algorithm experiments can be seen here:

- TFIDF:-
  Train acc – 0.60
  Validation acc – 0.59
  Test acc – 0.59
- GloVe:-
  Train acc – 0.61
  Validation acc – 0.60
  Test acc – 0.60

7. Experiment 4 (Hyperparameter Optimization)

The experiment 4 is based on using the best techniques from previous experiments and then tweak the hyperparameters to tune it.

7.1. Generalization

The same methodology is used as in 6.3 but this time, hyperparameters are being going to change.

7.2. Variation A (100 epoch and batch size -32)

Same methodology from 6.3 is going to be used. The change is in number of epochs and batch size. These two are being tweaked because, epoch is just like iteration, i.e. the number of times the model is exposed to the training

data, the more it is shown the more it will fit onto it, more epochs make the model learn complex patterns. While batch size is like how many samples should be put in mini batch such that the gradients are updated after the processing of one mini batch, this process allows to converse faster to minima, also, it can avoid local minima and go towards the global minima. The smaller the batch size the better the model will perform on the dataset.

In this variation, 100 epochs are being used while batch size is taken as 32. The results can be seen in Figure 19.

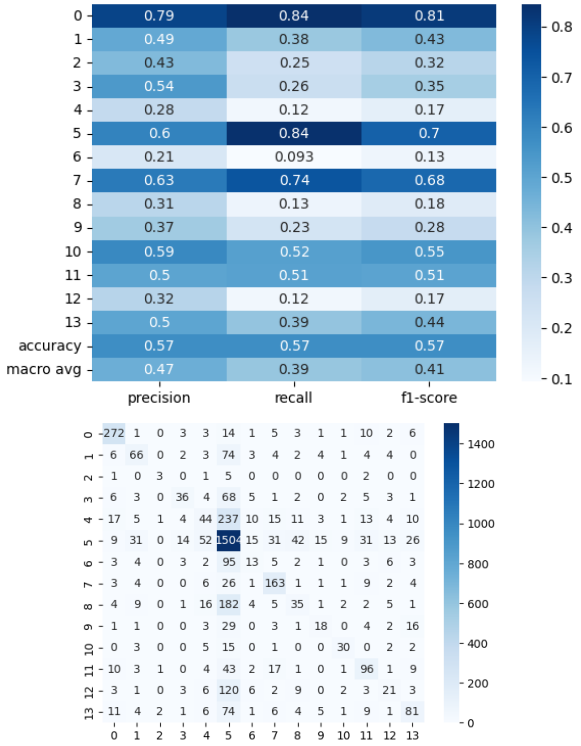
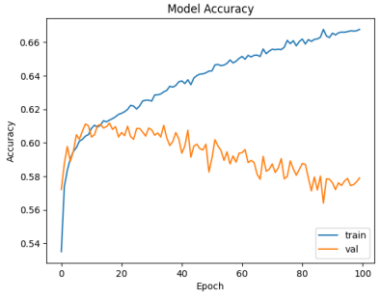


Figure 19: 100 epoch and batch_size 32 on BLSTM

It can be seen from this variation that the model has generalized well on the classes which were being predicted wrong the last experiment. As in Figure 18 and Figure 19, the f1 score for the labels 2,4,6,8 and 12 has been increased which is a big improvement. This shows that small batch size and giving more time to model can generalize the model better. But the model is facing high variance as its validation loss is increasing overtime which can be seen in Figure 20.



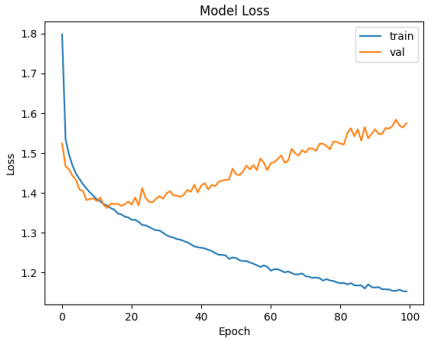


Figure 20: Accuracy and Loss graph for 100 epochs

7.3. Variation B (Learning rate and Dropout)

The most crucial hyperparameter in any machine learning task is the learning rate of optimizer. It controls basically how long/big the step is going to be to update the weights. On the other hand, dropout is a regularization term which helps the model to not overfit on the training data.

The default learning rate which was being used in all the experiments was 0.001 i.e. 1e-3, in this step the learning rate to be used is 3e-4, which will make the learning slower and gradual but the model will fit on the data well. To balance out that dropout term is given after BLSTM layer the dropout parameter is set to 0.3. The batch size is again 32 and 100 epochs. The results of the model can be seen in Figure 21.

This time the model has shown good performance, not overfitting neither underfitting, which can be seen in Figure 22.

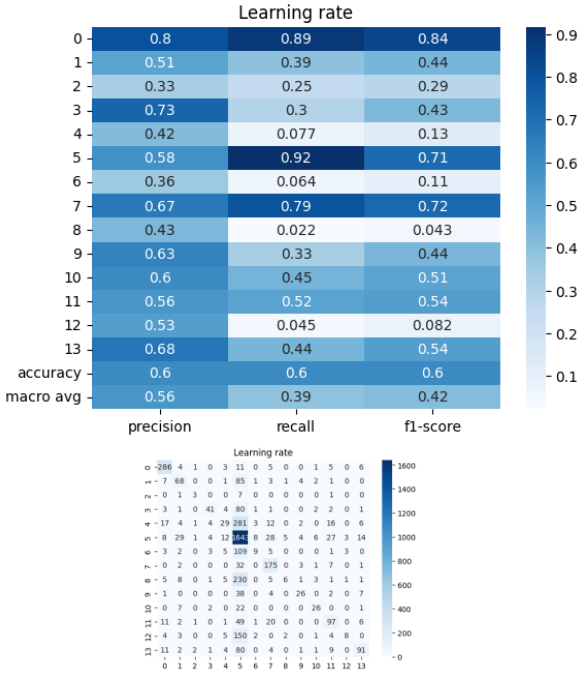


Figure 21: Learning rate change on BLSTM

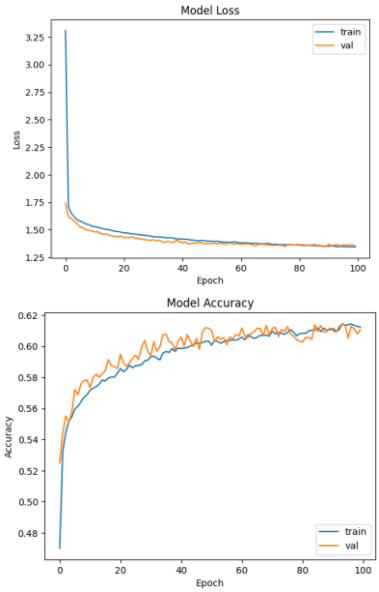


Figure 22: Accuracy and Loss graph for learning rate change

The f1 score for almost all of the labels have been seem to increase, also the model is not overfitting on the data which is a good sign. Again 2, 4, 6, 8, and 12 labels are being predicted wrong. While 0, 5 and 7 is being predicted right.

7.4. Variation C (Implementation of CNN on BLSTM)

The BLSTM are working fine, but we can improve it more by adding CNN layers on it. In the paper [5], 2D Max Pooling has been used which will sample better features than 1D pooling. They experimented using different models, and this technique outperformed RNNs, CNNs and also normal LSTM.

The hyperparameters and model architecture can be seen in Figure 23.

```
model = Sequential()

model.add(Embedding(input_dim=num_words, output_dim=100, input_length=maxlen,
                    embeddings_initializer=Constant(embedding_matrix), trainable=False))

model.add(Bidirectional(LSTM(units=100, dropout=0.2, activation='tanh',
                             recurrent_activation='sigmoid', use_bias=True,
                             recurrent_dropout=0, unroll=False,
                             kernel_regularizer=regularizers.l2(0.01), return_sequences=True)))

# Add convolution layers
model.add(Reshape((-1, 100, 1)))   # reshape output of BLSTM layer to 3D tensor
model.add(Conv2D(filters=128, kernel_size=(3, 100), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 1)))
model.add(Conv2D(filters=64, kernel_size=(3, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 1)))

model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(units=14, activation='softmax'))
```

Figure 23: BLSTM-CNN model

The BLSTM-CNN performed well and gave us better f1 scores than the previous model, which was fitted so well. BLSTM-CNN didn't fit well with this hyperparameters, still performed better. This shows that the CNN and MaxPooling works well with text data as written in the paper[5]. The results of this can be seen in Figure 24. The accuracy and loss graphs are give in Figure 25.

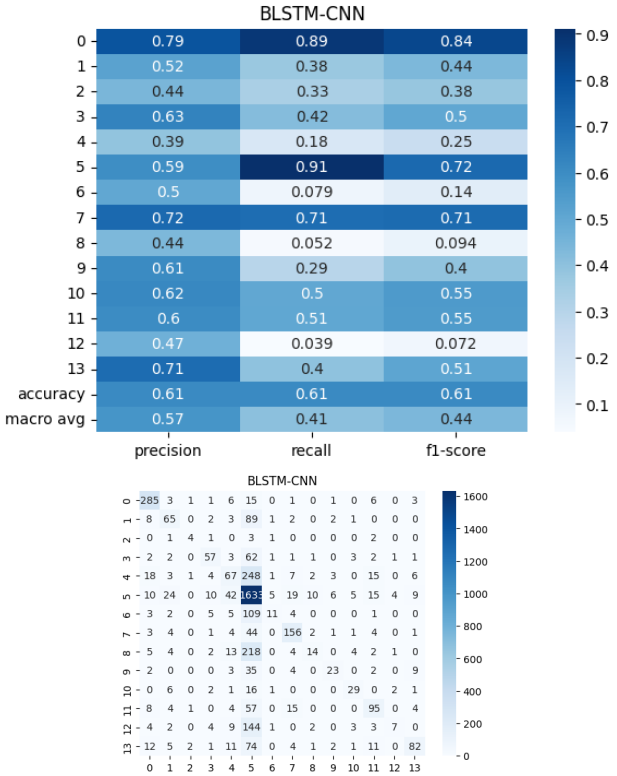


Figure 24: BLSTM-CNN model results

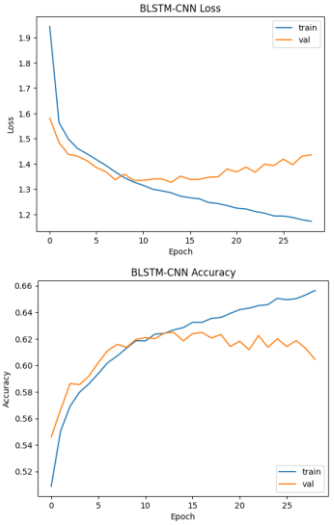


Figure 22: Accuracy and Loss graph for BLSTM-CNN

7.5. Verdict

This experiment consisted only the hyperparameter tuning. It showed how learning rate, number of epochs, batch size and regularization terms can help us to tune the model better to the data. The first variation of 100 epochs was having overfitting which was then fixed with tuning

learning rate and introducing dropout layer.

A implementation of CNNs was done after BLSTM in the third variation which gave the highest average f1 score of 0.44 making it the best model.

The results of Algorithm experiments can be seen here:

- BLSTM 100 epoch:-
  Train acc – 0.66
  Validation acc – 0.58
  Test acc – 0.57
- BLSTM learning rate:-
  Train acc – 0.63
  Validation acc – 0.60
  Test acc – 0.60
- BLSTM-CNN:-
  Train acc – 0.65
  Validation acc – 0.62
  Test acc – 0.61

8. Best Results

With all the experiments done, the following results have been obtained from hyperparameter tuning on these models, as followed in Table I.

Algorithm	Hyperpara. Tuned	Train Acc	Val Acc	Test acc
BLSTM	Epochs & Batch_size	0.66	0.58	0.57
BLSTM	Learning rate	0.63	0.60	0.60
BLSTM-CNN	Early stopping	0.65	0.62	0.61

Table I: Results.

Thus we can say that BLSTM-CNN was the best model and out of all these with these hyperparameters as in Table II. BLSTM-CNN was also used using early stopping which ran only for 26 epochs compared to 100 epochs of others.

HYPERPARAMS	VALUES
BLSTM UNITS	100 (Dropout 0.2)
BLSTM (ACTIVATION/RECURRENT ACTIVATION)	Tanh/sigmoid
BLSTM KERNEL REGULARIZER	0.01
CONV 2D (1)	100, (3,3) ‘relu’
MAXPOOL2D (1)	(2,2)
CONV 2D (2)	64, (3,3) ‘relu’
MAXPOOL2D (2)	(2,2)
OUTPUT LAYER ACTIVATION	Softmax
OPTIMIZER	Adam (0.0003)
LOSS FUCNTION	sparse_categorical_crossentropy
EARLY STOPPING	Patience = 15
BATCH SIZE	16

Table II: Hyperparameters of BLSTM-CNN

9. Conclusion

Based on the results obtained from all the experiments, the model is performing well taking in consideration that the data was imbalanced as well as it was only from one source i.e. reddit comments. If the question was if this model will predict the labeled emotions which were chosen on an unseen dataset, which might be totally different from this, then no, it might not fulfill its purpose to accurately detect the emotions. But, as our accuracy has been obtained as 61 percent on test set, it could be improved by adding regularization and training it on more epochs. Also, it was seen in the results that many emotions like realization, approval, disapproval, confusion and pride were poorly classified and were mostly predicted as neutral labels. While neutral, gratitude and love were predicted mostly correct due to having some impactful words like, thanks, love, glad, favourite and many more.

The model did achieve a good accuracy of 61% but the model was tested on a similar type of dataset which makes it biased towards the dataset. If the same model being tested on other dataset might get more or less test accuracy, but a good enough accuracy cannot be defined as in many cases like binary classification, it is generally suggested to get high accuracy, but whereas this project was having 14 labels that too with misbalanced dataset, which makes accuracy of about 80-85 as really good.

With the model we designed from inspiration from BLSTM-2DCNN[5] it performed well on the dataset, but there could be many thing done to improve it. The data being not balanced, had very less samples for labels like pride, fear, desire, etc while on contrast, some labels having exponentially huge amount of data when compared with it such as 12k in case of neutral. The problem of this was just solved by undersampling the high sample labels. But the performance can be improved by oversampling of labels. While class weights might also be used so that we can rank the classes having low samples be having high rank.

One more thing can improve the accuracy of the model, to club the similar labels from the 27 labels such that similar emotions are taken as one emotion, for eg: excitement, joy, amusement. Which will not only increase the number of samples but also generalize on similar type of emotions which will increase the accuracy and f1 score of the model.

References

[1] Thurnhofer-Hemsi, Karl, et al. "Radial basis function kernel optimization for Support Vector Machine classifiers." arXiv preprint arXiv:2007.08233 (2020).

[2] Basu, Atreya, Christine Walters, and M. Shepherd. "Support vector machines for text categorization." 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the. IEEE, 2003.

[3] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.
[4] Adamuthe, Amol C. "Improved text classification using long short-term memory and word embedding technique." Int J Hybrid Inf Technol (2020).
[5] Zhou, Peng, et al. "Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling." arXiv preprint arXiv:1611.06639 (2016).
[6] https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
[7] https://colab.research.google.com/drive/1HY-uYzLbEKgetOFRxAFqUWbxqpjnwdnY?usp=sharing

link to colab notebook:- COLAB